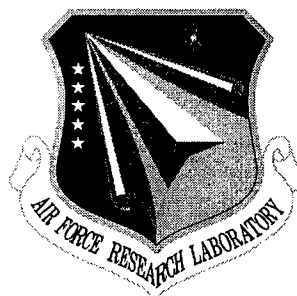AFRL-IF-RS-TR-2001-115
Final Technical Report
June 2001

CAPTURING THE EFFECTS OF C4I IN A CAMPAIGN CONTEXT: A PRACTICAL APPROACH TO CALIBRATING ANALYTICAL SIMULATIONS

Emergent Information Technologies, Inc.

Gregory Jablunovsky, Garth Morgan, Millard Barger, Joseph Krupp, Glenn Southan, and Clark Dorman
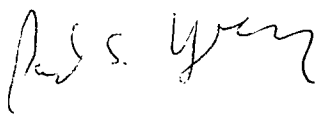
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
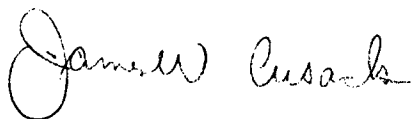
20010808 120

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-115 has been reviewed and is approved for publication.

APPROVED: PAUL S. YAWORSKY
Project Engineer

FOR THE DIRECTOR: JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFSB, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2001	Final Jun 99 - Dec 00

4. TITLE AND SUBTITLE

CAPTURING THE EFFECTS OF C4I IN A CAMPAIGN CONTEXT: A PRACTICAL APPROACH TO CALIBRATING ANALYTICAL SIMULATIONS

6. AUTHOR(S)

Gregory Jablunovsky, Garth Morgan, Millard Barger, Joseph Krupp, Glenn Southan, and Clark Dorman

5. FUNDING NUMBERS

C - F30602-99-C-0053
PE - 62702F
PR - 459S
TA - BA
WU - 93

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Emergent Information Technologies, Inc.
2600 Park Tower Drive, Suite 900
Vienna VA 22180

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFSB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2001-115

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Paul S. Yaworsky, IFSB, 315-330-3690

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The Air Force modeling and simulation community needs improved capabilities for measuring the effectiveness of command and control (C2) networks and processes in campaign-level analyses. A major modeling problem is capturing complex relationships between C2 network states and performance in a manner that is both traceable to empirical evidence and available in the timeframe required by an analytical simulation. Neural network technology offers a model abstraction technique with the potential to meet these criteria. Here campaign-level cause and effect relationships are captured using a custom neural network to help determine the effects of C2 network states on military operations. This neural network sub-model was then integrated into the Air Force's THUNDER simulation as a proof-of-concept. The resulting simulation showed sensitivity to state changes as provided by the neural network. In contrast to other C2 abstraction techniques, this neural network implementation was more credible because its values were directly derived from, and therefore more clearly traceable to, source data.

14. SUBJECT TERMS

Campaign Simulation, Campaign Analysis, Model Abstraction, Neural Networks, C4I (Command, Control, Communications, Computers, and Intelligence), C2 (Command and Control), Time Sensitive Targeting, Time Critical Targeting

15. NUMBER OF PAGES

40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

INDEX OF FIGURES AND TABLES

1.0 Abstract

The Air Force modeling and simulation (M&S) community needs improved capabilities in measuring the effectiveness of command and control (C^2) networks and methodologies in campaign-level analyses. Here, we use C^2 to identify the superset of the defense information exchanges, business logic and systems, alternatively labeled or encompassed within Battle-Management, Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (BMC4ISR). A major problem to date has been in capturing the complex relationship between state and performance in C^2 networks in a manner that is both traceable to empirical evidence and available in the timeframe required by a simulation-supported analysis. Neural networks are a model abstraction technique with the potential to meet those criteria. With regard to traceability and credibility, a neural network appropriately trained on data reflecting the direct relationships between a range of "causes" and "effects" offers the potential to classify the relationship of points within that neighborhood. For this project, we developed training data which reflected cause-effect relationships between C^2 performance in terms of shortest time path (latency) from sensors to shooters as a function of the C^2 network state. We evaluated several neural network training methodologies with a wide range of parameter settings and determined that back-propagation provided sufficient accuracy and efficiency for our purpose. With regard to responsiveness, the neural net can (for points within its trained range) approximate relationships at a speed compatible with the run-time demands for campaign analysis; i.e., typically tens of days of simulation time in a few hours. For this project, we used values associated with cause-effect relationships within a campaign simulation to determine effects of C^2 network state on military operations. We integrated a specialized, custom neural network into the Air Force's THUNDER campaign-level simulation and evaluated its performance in a proof-of-concept. In the experiment, the simulation showed sensitivity to state changes supplied by the neural network. In contrast to previous techniques for abstracting C^2 effects for campaign-level models, the neural network was more credible because its values were directly derived from, and therefore are more clearly traceable to, source data. The results of the demonstration support the project's premise that neural networks have the potential to provide sufficient levels of credibility and responsiveness necessary for simulating and evaluating the effects of C^2 network state in campaign-level military operations.

2.0 Executive Summary

Decision-makers need insights into the dynamic interaction of complementary and competing systems in the context of overarching military objectives. Previous and current approaches to deriving aggregate C^2 effects from detailed performance models are either too slow to be practical or lack sufficient traceability to inspire confidence. Better representation of C^2 within campaign simulations will help Air Force analysts credibly quantify the impact of C^2 from a CINC-level, war-fighting perspective. Model abstraction provides an approach to provide this improved connection between C^2 network and operational dynamics.

Simulation of C^2 networks has historically emphasized individual system performance with little architectural context or credible linkage to "bottom-line" measures of combat outcomes. Renewed interest in modeling C^2 effects and relationships stems from emerging

network- intensive operational concepts, information operations and other warfighting innovations and challenges. This demands improved methods to span the analytical hierarchy between C^2 system performance models and theater-level models. Neural network technology offers a modeling approach that can abstract the essential behavior of high resolution C^2 models within a campaign simulation. The proposed methodology uses off-line learning of the relationships between network state and campaign-impacting performance of a complex C^2 architecture and then approximates that performance as a time-varying parameter in an aggregated simulation. Ultimately, this abstraction tool offers an increased fidelity of C^2 system simulation that captures dynamic network dependencies within a campaign context.

We investigated a neural network to quickly and accurately derive inputs for campaign-level simulations from detailed model outputs. Neural networks are loosely modeled after the networks of neurons that make up the brain, and are capable of learning input-output mapping and generalizing the relationships between them. We selected and trained several types of neural networks on results obtained from more detailed models. The neural network approximations were tested against the detailed model results to determine the best (in terms of speed and accuracy) candidate for incorporation into a campaign-level simulation. We identified an area within a campaign-level simulation where C^2 effects are explicitly represented but for which values are currently estimated. The selected neural network was integrated into the campaign simulation to receive appropriate inputs and dynamically generate outputs as the campaign progressed. We used the integrated neural network/campaign simulation in a demonstration application to evaluate the analytical utility of the tool for assessing the military worth of C^2. Theoretically, the current methodology in THUNDER could produce results on a par with the neural network, but this would require significantly more effort on the part of analysts to determine the appropriate distribution parameters by manually aggregating the results of detailed models or other source data into the appropriate distributions for THUNDER input. The neural network, by comparison, can more quickly support the existing logic by abstracting from a range of values reflecting appropriate campaign relations , drawing from data more directly derived from operational experience or more detailed models. In contrast to previous techniques for abstracting C^2 effects for campaign-level models, the neural network can be designed to be more credible because its values are directly derived from, and therefore are more clearly traceable to, source data.

Our results with simulation applications for neural networks substantiate the possibility of significant improvement in the analytical depth and subsequent credibility of simulation-driven exploration of C^2 systems and capabilities. Enhancements in cross-model calibration techniques (those that use results from the output of a detailed model or simulation directly or to develop input for a more aggregate tool) or model abstraction for C^2 issue areas will address a long-standing need across the defense modeling community as a whole and the Air Force in particular.

3.0 Background

The US Air Force faces difficult decisions as it strives to develop and acquire C^2 technologies and systems in an era of doctrinal change, operational evolution, technical advances (e.g. processing and network) and constrained resources. Today's acquisition environment is a

complex trade space involving the anticipated costs and benefits of vastly dissimilar elements. Justifying developmental funding and acquisition authority in this environment makes it more necessary than ever to objectively quantify the relative contribution of a system or capability to higher-level systems within the military infrastructure and, ultimately, to outcomes of major military operations.

Particularly as the Air Force attempts to evaluate force structures and operational concepts implicit in an information-based approach to combat, it will require analysis extending far beyond mere relative performance differences between alternatives within a common functional class. Instead, decision-makers need better insight into the dynamic interaction of complementary and competing systems in the context of overarching military objectives. For example, in terms of effects on military operations spanning weeks and months, what are the implications of fluctuations in communication delays measurable in nanoseconds or minutes for a given C^2 network topology? These insights help frame compelling answers to the "so-what" questions posed by the Joint Requirements Oversight Council (JROC), Congress and USAF's internal planning and budgeting processes.

Decision-makers have traditionally looked to campaign-level simulations to illuminate issues of comparative military utility at the operational level of war. Such analysis, however, is given greater credibility when the aggregate effects of a given system or capability at the high end of the modeling hierarchy (campaign level) can be derived from, or are traceable to, lower-level, more detailed models (mission, engagement and engineering levels). This is certainly the case for "shooter" systems. Well-established, formal processes exist to "translate" tactical performance detail into lethality and survivability values that drive probabilistic campaign-level representations of cumulative attrition effects. In the Air Force's THUNDER campaign analytical simulation, for example, the SABSEL process is the means by which THUNDER's air-to-ground probability of kill (P_k) values are calibrated to data contained in the Joint Munitions Effectiveness Manual. Similarly, THUNDER's ground war representation employs the Army's ATCAL (Attrition Calibration) process to provide a traceable analytical thread from the test range through engagement-level results (captured via the Combat Sample Generator model) to aggregated outcomes. For attrition-related areas, the calibration task benefits from years of research and study. These processes are founded on empirically based and consensual understandings of quantifiable physical effects. This relatively transparent translation or "calibration" procedure imparts a significant level of credibility to simulation-driven evaluations of the campaign contributions of firepower-related systems. Unfortunately, while C^2 systems and capabilities are indisputably of critical and increasing importance to the successful conduct of U.S. military operations, a calibration process analogous to those providing the engagement lethality parameters outlined above has proven elusive. The Military Operations Research Society's C^4ISR Working Group concluded that the analytic linkage between C^2 architecture and measures of effectiveness is weak.[1] The modeling and simulation (M&S) community has yet to capture a credible representation of cumulative C^2 effects on theater-level operations in support of acquisition trade-off decisions.

One reason for this is the very nature of the relationship between entities in an interconnected architecture. The complex network of relationships in a C^2 grid contrasts with the more linear and easily understood relationships between shooters and targets. Lacking as a result

3

are the stable underpinnings for effective abstraction methodologies to translate variances in C^2 performance observable at the tactical level into cumulative effects relevant at the operational level. To date, the most common approach to representing C^2 in campaign-level simulations has been an implied, essentially binary solution. For example, given a "good communications state," information gets through and actions occur as planned. When the "communications state" is "bad," information does not go through and decisions and actions are delayed. While the simulation methodology is certainly crude, its utility is limited more by the analytical challenge of identifying sources for the values reflecting state changes. Under what circumstances, for example, does the "communications state" result in a delay of high-priority message traffic? What is the scope and extent of that delay across a theater? Is it uniform or varied? The current approach is "off-line calibration;" essentially an ad-hoc extrapolation where subject-matter experts decide whether or not the results "look about right." While certainly feasible, this approach affords shallow analytical depth for exploration and explanation. Consequently, the perceived credibility of the results suffers accordingly. Model abstraction provides an approach to capture the essential characteristics of a richer description of C^2 relationships within an aggregate model.

From a technological perspective, detailed models certainly exist to identify and evaluate performance parameters for C^2 technologies and systems. However, merely demonstrating that System A with its faster baud rate can handle x-many more messages per minute than System B does not establish its relative value to the war-fighter. Campaign-level simulations are appropriate tools for that task, but current calibration approaches fall short in several ways.

For certain areas of interest similar in complexity to C^2, such as air-to-air and surface-to-air interactions, calibrating campaign-level effects currently imposes a significant burden on resources. Typically, higher-level models are calibrated by scaling the outcomes of their events and interactions to match specific case data generated previously by detailed models. For example, analysts compare specific THUNDER interactions to the results of numerous BRAWLER or SUPPRESSOR runs in an effort to "tune" the campaign-level simulation outcomes to match the detailed model results. Accurate scaling requires a comprehensive library of detailed model results. While satisfactory in theory, the time and effort necessary to populate an exhaustive, full-factorial library of detailed results is insurmountable in practice. Given the stochastic nature of these detailed models and the number of variables, the number of cases and replications possible borders on the infinite. A variant of this approach involves minimizing the number of cases by "surrogating" (e.g., an F-16 C is referenced for airframes performing similar roles and missions). However, surrogating can only be taken so far before credibility again becomes an issue.

Another approach that has been attempted for calibration of C^2 relationships between models is the federation (via the High-Level Architecture-Run Time Interface, or HLA-RTI) of models of varying resolution. This approach has the appeal of promising the highest fidelity for each model of interest, but there are practical drawbacks. First, each federation requires a high level of expertise in terms of data collection, formatting and model execution. Furthermore, a federation runs only as fast as its slowest member. For example, in the case of detailed C^2 models like NETWARS with greater-than-real-time runtimes, this would be a significant impediment to federating with a campaign-level model that requires 90 days for a single

4

replication. Federations attempting to incorporate communications, ISR and intelligence "fusion" models within a campaign-level simulation have heretofore involved relatively small and brief scenarios. In practice, approaches attempting to examine large-scale, extended duration combat scenarios by federating a campaign-level simulation with detailed C^2 models will either suffer from impractical runtimes or employ artificial checkpoints (time steps) that unduly influence the results.

Better representation of C^2 within campaign simulations will help Air Force analysts more credibly quantify and evaluate the C^2 impact from a CINC-level, war-fighting perspective. The challenge lies in the timely and credible (traceable) calibration of campaign-level effects from detailed model outputs in a manner that supports computer-based analysis across multiple levels of abstraction.

4.0 Project Description

4.1 Precepts of the Approach

At the conceptual level, we envisioned addressing the C^2 calibration problem by means of a neural network employed as an abstraction mechanism that would supply a campaign level military simulation with credible, empirically-derived values related to C^2 performance. Neural networks are modeled after the network of neurons that makes up the brain and is capable of learning input-output mappings and generalizing relationships between them. Because they can be designed to model diverse input-output relationships without foreknowledge of the characteristics of the input-output source, neural networks have been used in a variety of contexts requiring automated recognition and approximation techniques. A supervised neural network is trained or "taught" on pairs of input-output data that ideally span the range of the problem space. Given inputs within that range after learning, including inputs that the neural network has not seen before. The neural network is theoretically able to approximate the appropriate output by recognizing input patterns with their output mapping and generalizing beyond those inputs already learned.

This research project proceeded from the premise that neural networks could address the shortcomings in practicality and traceability evinced by previous and current calibration approaches. The speed of a neural network promised to overcome the run-time problems inherent in a federation approach. The accuracy of neural network approximation promised to eliminate the need for exhaustive results libraries for a near-infinite set of variables and conditions. Another advantage of neural networks is particularly appropriate in the context of verification and validation (V&V). Credibility of analytical results, as the by-product of V&V, is a concern when the availability of empirical data from lower-level models is not necessarily matched by our understanding of the underlying behaviors and relationships among the various entities involved. Neural networks do not require explicit representations of relationships between variables in order to approximate them. This feature makes NN's particularly attractive for representing complex C^2 network phenomena.

We selected THUNDER to serve as the host simulation, with a neural network functioning as an abstraction mechanism. Sponsored by the Air Force Studies and Analyses Agency (AFSAA), THUNDER is the principal campaign-level analytical tool in the Air Force Standard Analysis Toolkit and is currently used for force structure studies and analyses of alternatives (AOA) across the Air Staff. We believed THUNDER to be an ideal experimental platform within which to demonstrate the concept of neural networks serving as an abstraction device. With a pedigree extending back more than 17 years, the internal dynamics of THUNDER as a campaign model are well documented and input to output relationships are well understood. C^2 performance is characterized by the volume, quality, speed and accuracy of information transmission and processing. While C^2 performance is a factor in a wide range of the actions and processes that comprise a military operation, for the purposes of this experiment we focused on the impact of variations in C^2 performance within a single specific "thread" or operational mission area.

Proceeding from an extensive understanding of the structure and characteristics of THUNDER as it exists in its current release (Version 6.7), we chose the operational mission area of Offensive Theater Ballistic Missile (OTBM) as the thread most likely to provide the clearest perspective on the practicality and analytical utility of this abstraction tool. OTBM missions involve the use of air and space sensors detecting either movement of TBM transporter/erector/launchers (TELs) or actual missile launches to cue strikes by orbiting on-call aircraft. The SCUDCAP missions from DESERT STORM are an historical implementation of the OTBM concept. Because this mission area involves time-critical targets, C^2 performance for the purposes of this demonstration was denominated in terms of latency or delay, i.e., the total elapsed time between the generation of data by a sensor, the processing of that data into useful information, the transmission of that information as a message across a network architecture and the ultimate reception of the message by a shooter.

4.2 Specifics of the Approach

Figure 1 illustrates the practical elements, connectivity and data flows associated with integrating a neural network into THUNDER to provide traceable linkages to C^2 performance values. First we developed a data set representing the relationship between a range of changes in C^2 network states (input) and resulting C^2 performance (output) on which to train a neural network. We generated a set of data reflecting the performance of a given architecture across a range of conditions. Working with a software implementation of a neural network developed as part of a toolkit under a research effort to explore the application of Genetic Algorithms to the allocation of airpower to missions in THUNDER, we evaluated various training methodologies to select the most appropriate in terms of accuracy and speed. Many methods of training the neural network through back-propagation have been developed.[2] The quickest and most appropriate methodology for a particular application depends on the input-output relationships that need to be modeled by the network. We then developed the software interfaces necessary to embed the developed neural network into THUNDER and implemented the modifications. Finally, we developed scenario data for THUNDER to establish the simulated operational conditions that would demonstrate sensitivity of the model to state changes in the C^2 network architecture.

6

Neural Network/THUNDER Integration Architecture

Figure 1: Elements, connectivity and flow of the project.

4.2.1 Training Data Development

The original concept, described in the original white paper submitted under the Broad Agency Announcement, for this task as a subset of the effort envisioned the collection or production of a training data set from the output of detailed models reflecting the performance of "real-world" C^2 architectures as a function of state conditions. This would have involved many excursions of a complex network architecture model, with associated data post-processing and analysis to arrive at a training set of the requisite quality for actual analytical use. This, however, was beyond the scope of the funded effort. Consequently, to support a proof-of-concept demonstration, we generated an illustrative training set upon which to train the neural network.

We found that existing data in the C^2 community generally focused on the performance of a healthy, as-is architecture with excursions describing improvement from the addition of new or enhanced systems. This served as a good source for describing baseline network performance, but provided no corresponding information that explicitly described how the performance changed as a function of network state. This precipitated the need to translate a set of network state variables to a corresponding end-to-end latency which would serve as the training data set for our neural net implementation.

7

To generate this training set, we constructed a notional C^2 architecture analogous in scope and scale to real-world architectures that support prosecution of TBM targets. The Combat Air Forces provide a well-documented concept of operations.[3] Joint doctrine provides the context for the architecture and outlines operational guidance.[4] These served as the principal source documents for establishing the structure of the architecture. Then we determined appropriate latency values for the various links in the architecture. Table 1 and Figure 2 describe the elements and connectivity, respectively, of a notional architecture to illustrate the general approach and highlight the sensor-to-shooter connectivity.

Table 1: Elements in the C² Network Architecture.

Note: These systems represent some aggregation of functionality. Each is named to define the dominant system in the operational process and is described the as capability of that system and some associated sub-systems.

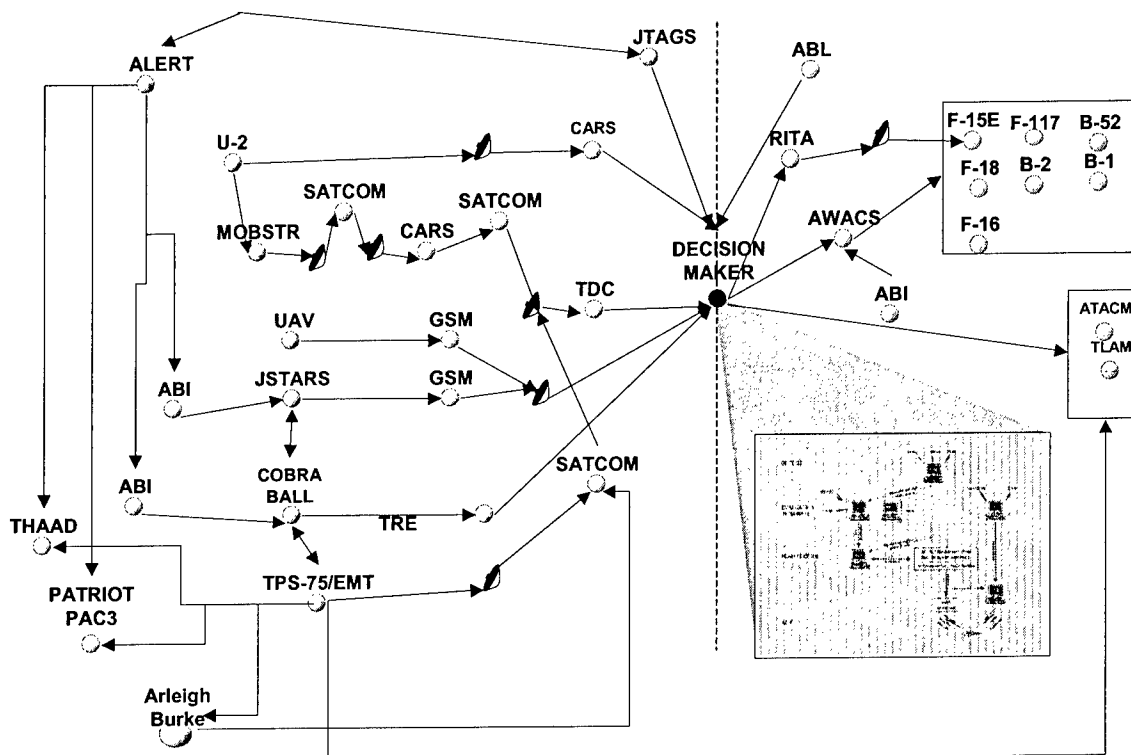| System	Description
ARLEIGH BURKE-class platform	Guided missile destroyer with AEGIS defensive System. Cooperative Engagement Capability
Airborne Broadcast Intelligence (ABI)	Makes TIBS/TRAP broadcast messages available to airborne platforms. Useful for sensor cueing and situational awareness.
Airborne Laser (ABL)	Boost-Phase-Intercept weapon platform with onboard sensors, directed-energy engagement capability and launch-point/impact-point prediction
Attack and Launch Early Reporting to Theater (ALERT)	AFSPC processing node for launch data Correlates and analyzes sensor and intelligence data in CONUS for transmission to theater forces
Army Tactical Missile System (ATACMS)	Deep, indirect fire weapon launched from MLRS. Component of TCT prosecution through Army Battlefield Coordination Division in Air Operations Center
AWACS (E-3)	Airborne C² node conducts battle management in attack-operations mission. Controls fighter to target. Here, also logically represents theater JTIDS network
B-1, B-2, B-52	Bomber platforms. Expeditionary Force experimentation has demonstrated feasible role in prosecuting TCTs
Cobra Ball	Detects and Tracks Infrared targets, launch-point/impact-point prediction, multi-spectral sensor, cue for other sensors
Contingency Airborne Reconnaissance System (CARS)	Ground processing and control station for U-2 sensor suite
Decision Maker	Aggregates the TCT functions Although decentralized decision authority has been demonstrated in airborne nodes, this architecture represents centralized authority at a ground-based C² node
Defense Support Program(DSP)	Satellite boost phase detection and tracking
Expert Missile Tracker(EMT)	Provides track correlation and fusion along with launch-point, impact-point prediction. Associated with TPS-75.
F-15E	Attack-ops. weapon system
F-16	Attack-ops. weapon system
Ground Station Module (GSM)	Ground station for processing (and possibly control of) airborne sensor data.
Intel Broadcast System	
JSTARS	Detect, locate, track, classify moving targets (Moving Target Indicator)Detection, identification and targeting of TCTs capable of tracking Transporter/Erector/Launchers (TELs) to/from hide and reload sites
Joint Tactical Ground Station (JTAGS)	Receives DSP and SBIRS data directly in theater. Capable of relaying processed, real-time attack warning information throughout theater
JWICS	1. The Sensitive Compartmented Information (SCI) portion of the Defense Information System Network. It incorporates advanced networking technologies that permit point-to-point or multipoint information exchange involving voice, text, graphics, data, and video teleconferencing. 2. A high speed (T-1, 1.544 Mbs) communications system designed to provide secure, system high (Top Secret/SCI) data, interactive video teleconferencing, and video broadcasting capabilities to its subscribers.
MOBSTER	Ground relay station for U-2 when operational orbit and ground station (CARS) are beyond Line-Of-Sight
RITA	Communications van capable of pushing properly formatted imagery to attack asset cockpit to facilitate target acquisition
Space Based Infrared System (SBIRS)	Surveillance platform constellation providing missile launch detection, launch point/impact point prediction. (Low) Midcourse and re-entry tracking and discrimination
Tactical Data Dissemination System (TDDS)	Network for distributing intelligence data
Tactical Receive Equipment (TRE)	Equipment suite for receiving intelligence broadcast messages (TDDS)
Joint Tactical Information Distribution System (JTIDS)—	TDMA network for TADIL-J (Link-16) data link and IJMS messages. Provides secure integrated communications, navigation, and identification capability for application to military tactical operations.
Theater Deployable Communications(TDC)	Integrated comm. package that serves to connect in-theater C² nodes together and reach back to CONUS over SATCOM
TPS-75	Ground-based radar for area air defense. Associated with EMT
U-2	Associated radar sensors provide Moving Target Indicator (MTI) information and images.
UAV	Generic unmanned sensor platform. Here, assumed to carry imaging sensor.
Tactical Land Attack Missile (TLAM)	Tactical Land Attack Missile Sea-launched weapon with potential for re-targeting data link by 2010
UHF Voice	Line-of-sight tactical comm. for battle management

Figure 2: C^2 Network Architecture Connectivity.

We represented the C^2 network (nodes, connectivity and link latencies) as a weighted directed graph. This approach is in line with typical C^2 network node connectivity diagrams from C^2 architecture descriptions as referenced in guidance published by DoD C^4ISR Architectures Working Group.[5] One benefit of this approach is that it provides a mathematical formalism for describing a network, allowing the use of shortest-path algorithms such as Dijkstra's to determine minimum latencies associated with different network configurations.[6]

We used this weighted directed graph / Dijkstra's algorithm approach to establish a relationship between the state of the C^2 network and origin-to-terminus "path length" representing the aggregate delay. In our case this value corresponds to a sensor-to-shooter latency. This approach necessitated that we assume each link in the network could be represented by a single constant latency. This implies infinite bandwidth (ignoring traffic congestion effects and the "size of the pipe") and no parallel links between C^2 nodes (ignoring redundant connections).

We wrote a Java program to take a given C^2 architecture and find the shortest path using Dijkstra's algorithm. Using this program, we computed the total sensor-to-shooter latency for the large number of analytically useful network configurations corresponding to combinations of nodes missing from the network. This data mapped C^2 network states to performance for a large number of possible C^2 network configurations. These configurations correspond to network state

10

changes associated with attrition (destroyed or damaged assets), interference (degradation associated with jamming) or availability (deployment).

4.2.2 Neural Network Software

Neural networks are loosely modeled after the network of neurons that makes up the brain and is capable of learning input-output mapping and generalizing relationships among them. Because they can accurately model diverse input-output relationships without foreknowledge of the characteristics of the input-output source, neural networks have been used in a variety of contexts for recognition and approximation. The neural network is trained or "taught" on pairs of detailed input-output data that span the range of the problem space. Given inputs within that range after learning, including inputs that the neural network has not seen before, the neural network is theoretically able to accurately approximate the appropriate output.

During learning, the neural network is presented with an input pattern corresponding to the input values of a higher resolution C2 network model. The inputs to the neural network cause activation at the top, or input, layer of artificial neurons within the network. Through weighted connections with other nodes, the active nodes cause activity to increase or decrease at nodes throughout the network. After several layers, the activities of the bottom, or output, nodes represent a calculation that depends on the input values and weights of connections of the nodes. The output activities can be compared to the output produced by a high-resolution model; the difference between them represents an "error" in the processing of the neural network. The size and direction of the error is used to modify the weights of connections within the network in a process called "back propagation." After several trials of input presentation and error back propagation, the error size goes down and the network is said to have learned that input-output pattern. Importantly, a single network can learn many such input-output patterns and, when presented with input patterns that are between learned patterns, the network will interpolate between the learned outputs. Mathematically, it has been proven that a three-layer neural network is capable of approximating any input-output function to a desired level of accuracy. [7]

In each artificial neuron in the network within the hidden layers, the weighted input signals are summed, and converted to an output. The total input to a neuron is the sum over all the neuron inputs connected from the previous layer times their weights. Once the total input to the neuron is computed, it is processed by the neuron through its non-linear "transfer –function." In this project, the standard sigmoid function is used for the transfer function, which "squashes" the inputs to an output value between 0 and 1.

During training, the NN is presented with a given input pattern. The input neurons receive the inputs and produce output for the second layer. The neurons in the second layer each process their inputs and produce outputs to the output layer. The output layer also processes its inputs and produces the network's output. While the network is being trained, there is a known output that the network is supposed to produce. The known output is compared with the network's output, producing an error value. Based on the error size, and the sizes of the inputs from the intermediate layer neurons, the weights of the arcs from the intermediate neurons to the output neurons is modified to reduce the error and an error is assigned to each of the intermediate neurons. Similarly, the error at the intermediate layer neurons is used with the weights of the arcs from the input layer neurons to adjust the weights of the arcs between the input neurons and the intermediate neurons.

The process of adjusting the weights of the arcs from the outputs to the intermediate layer to the input layer is called "error back-propagation" and is the standard present method of training. When there are many input-output pairs for the neural network to learn, they are presented one at a time, with the weights being adjusted slowly to adapt to all of the inputs. After all the inputs have been shown, the process is repeated as necessary. The final trained network is able to accurately reproduce the input-output pairs.

The training process continues until one of several criteria is met. The first criterion is that a certain number of repetitions (called "epochs) of all the input patterns has been presented to the neural network. The limit in our neural network implementation is 200 epochs of training and is intended to constrain the amount of time that the neural network trains. A second criterion is that the total amount of error (summed over all the input patterns) declines to an acceptable (low) level, indicating that the neural network has sufficiently learned the input patterns. A third criterion is that the rate of change in error falls within a specified bound, regardless of the total error. A small change in error indicates that the network has converged to a local minimum in the amount of error and will not benefit from further training.

The baseline neural network C++ code used for this effort was developed as part of a more basic artificial intelligence toolkit under a Small Business Innovative Research (SBIR) Phase II Project entitled "Development and Demonstration of a Genetic Algorithm and Neural Network Toolkit (GA/NNT) for Campaign Modeling." This work was sponsored by the U.S. Air Force Electronic Systems Center (ESC), was performed by System Simulation Solutions, Inc. (now Emergent Information Technologies, Inc.), and was funded under Topic AF96-029 of the SBIR Program. The application area within the SBIR work was to more broadly apply artificial intelligence technology to optimizing the allocation of air power resources to missions. The experience and basic algorithms developed here provided the point-of-departure for the adaptation and application of the neural network components of the toolkit to the C4ISR problem domain.

The neural network code as developed for the above-mentioned effort is an object-oriented implementation written in C++. The application's core is a fully connected, feed-forward network class upon which a back-propagation class has been built. The feed-forward class propagates the input-output computation according to data describing connection weights, neuron connectivity and activation functions. The back-propagation class provides the methods to adjust the weights according to errors between the computed output and training set known output. We implemented the neural network tools to handle an arbitrary number of inputs, outputs, nodes per layer, and of hidden layers. This neural network class is embedded in two application programs where instances let us implement neural network objects-- the Neural Network Trainer and the Neural Network Evaluator.

The Neural Network Trainer program is used to train the network from a given training set specified in an input data file. The resulting trained network is stored as an output data file. The user defines, through a configuration file:
- Which type of network training to use: normal back propagation, radial basis function, or batch learning back propagation;

- How to use the data set for training: number of samples to be used for training and percentage from the set to be used for testing;
- Which stopping strategy to use: stop when acceptable error is reached, stop when the change in the error is under a certain threshold, or stop after a predetermined number of iterations have passed;
- How many hidden layers will be used and how many nodes will be in each hidden layer.

The Neural Network Evaluator program is used to test the accuracy performance of a trained network against the original training data set. It takes as input a training set and a data file defining the trained neural network as output from the Neural Network Trainer. It runs each data point in the training set through the network and compares the output against the desired output from the training set. The program reports the number the net correctly classified, the number incorrectly classified, and the percentage correct.

4.2.3 Neural Network Training

Several neural network architectures and a wide range of network parameters were used during the exploratory phase of this project. Based on that exploration, an iteratively updated feed-forward, error back-propagation network was selected.

4.2.3.1 Training Methodology Exploration

This section discusses the different networks that were investigated during the exploratory phase and their relative performance.

The supervised, feed-forward class of neural network is suitable for our application given our requirement to model the input-output relationships that map C^2 network states to sensor-shooter latencies. In this class of networks, inputs are presented to the first layer, signals are forwarded to one or more hidden layers, and results are presented in the final, or output, layer. The network is trained to produce appropriate outputs by varying connection weights between nodes in the layers.

We considered four neural network training methodologies:
- Iterative error back-propagation;
- Batch error back-propagation;
- Batch error back-propagation with momentum; and
- Radial basis functions.

Each of these training methodologies is discussed briefly along with the results of our exploration. The back-propagation algorithms are based on determining the direction of weight change that will result in decreased weight (the local derivative) and using the chain rule to determine values for previous layers of nodes. The radial basis function network is based on doing a nonlinear transformation into a space in which the problem can be solved using linear algebra.

13

During supervised training, the goal is for the neural network to learn as closely as possible the correct relationships between inputs and outputs. The difference, or error, is one measure of the performance of the network. However, a more important measure of neural network performance is the correct classification of input-output pairs using data it has not been trained on. When used as an abstraction of the detailed C^2 network model within a more abstract model, the network will likely be presented with many novel input patterns that are not elements of the training set. We reserved a subset of the state-performance pairs from the training process, using these as a test set to evaluate the error in classifying the relationships on a C^2 network states outside the those used in training. This second measure gives us an idea of how well the network will accurately represent the C^2 network given those patterns. In addition, it is possible for a network to accurately learn input-output relationships and yet fail to generalize well.

The exploration training set contained 1,250 input-output pairs. These pairs were sampled from the comprehensive set of configurations corresponding to combinations of various operational nodes missing from the C^2 network. A test set of 125 pairs was set aside to determine how well the neural network would generalize. Each network was reset and trained on different sets of pairs selected from the total input space 10 times. In training the radial basis function network, the training set was presented once. However, in training error back-propagation networks, the input-output pairs were presented to the network repeatedly until the performance improved sufficiently to meet the established error criteria or until all were correctly classified. The presentation of all the input-output pairs is called an epoch and it can take several hundred epochs of training for the network to be satisfactorily trained.

4.2.3.1.1 Iterative Error Back-Propagation

During error back-propagation training, inputs are presented to the network. The information feeds forward through the network resulting in activities at the output nodes. The activity at the output nodes are compared to the correct outputs. Based on the difference, the weights within the network are changed accordingly. In practice, the weights between the hidden nodes and output nodes is modified based on the size and direction of the errors. The contributions of the hidden nodes to the errors are then used to modify the weights between those and nodes in the previous layer. This is referred to as error back-propagation.

In iterative back-propagation, the weights are modified after each input-output pair is presented to the network. In theory, this could lead to instability of the learning process because changes due to one input-output pair could impact performance of a different input-output pair. By training on conflicting individual pairs, the network could oscillate between values and fail to converge. In practice, however, this did not happen with our training set. We attribute this to observation to training step sizes, or the product of training speed and errors, that were small compared the distance between minima corresponding to solutions.

The adjustable parameters in our iterative error back-propagation networks were the number of hidden layers, the number of nodes per layer, and the training speed which represents the amount of weight change as a function of error size. During our exploration of the iterative error back-propagation approach, the number of hidden layers varied from 1 to 3, the number of

nodes per hidden layer varied from 10 to 50, and the learning speed was varied between 0.001 and 1.

Our investigation of the iterative error back-propagation network resulted in the following observations for this application:

- Classification performance did not improve by using more than one hidden layer. However, networks with additional layers required more training time because of the larger number of weights to be adjusted.
- Classification performance was optimum using approximately 30 nodes in one hidden layer. This is consistent with the heuristic that the number of hidden nodes should be between the number of input nodes (34, in our case) and the number of output nodes (18 for our neural network). With higher numbers of hidden nodes, the performance improved during training but declined during testing. This was a classic example of over-fitting.
- Classification performance was not highly dependent on the training speed between 0.0025 and 0.01. However, the network trained more quickly with higher values. A training speed of 0.01 was selected for later runs.
- The best performance, obtained with one hidden layer having 30 nodes, and by setting the training speed to 0.01, gave us correct classification for 92% of test input-output pairs.

4.2.3.1.2 Batch Error Back-propagation

In batch error back-propagation, the weights are not changed until all the training sets have been presented to the network. As alluded to above, this technique is intended to avoid the network from oscillating and failing to converge. With the training sets used in this project, the networks did not get caught in spaces where oscillations occurred, and therefore benefits of using batch error-propagation were not realized. Instead, because all weights are not changed until the end of an epoch, training occurred more slowly. The results for batch error back-propagation were:

- Overall performance was not improved over the iterative back-propagation results. Best performance remained around 92%.
- The optimum training speed was much lower, set between 0.0001 and 0.0025. This is because changes in weights were summed over a large number of input-output pairs. At higher values (>0.01) the network did not converge at all.
- Changes in other parameters, such as number of nodes per hidden layer and number of hidden layers, gave results similar to iterative error back-propagation.
- Training was much slower, taking 4 to 10 times as long as iterative error back-propagation.

4.2.3.1.3 Batch Error Back-Propagation with Momentum

A common variant of error back-propagation is to add a momentum factor to facilitate convergence. Using momentum, weight changes during each epoch are also affected by weight changes from the previous epoch. The idea behind this is that the momentum term can help carry the weights out of a local minimum. In addition, training speed often improves because the

momentum term evens out 'bumps' on the way to the global optimum. The additional parameter used here represents the value of the momentum term, which we varied between 0.1 and 0.9. A value of 0 would be the same as not using momentum.

Similar to the results of the batch process, performance was not improved over the iterative process. The speed was improved due to the momentum term, but not enough to overcome the advantage that the iterative process has over the batch process Specifically, the batch processes took smaller steps in exploring the solution space compared the iterative method. In summary, the results for batch error back-propagation with momentum were:
- Final classification performance similar to that of iterative and batch processing.
- Longer training time than iterative processing but shorter than batch.
- Optimum momentum value of 0.9 and other parameters similar to batch error back-propagation.

4.2.3.1.4 Radial Basis Functions

Radial basis function networks operate differently than error back-propagation networks. The network uses some of the training input-output pairs as known points in the space, called the basis. The rest of the training points are used to represent a surface in the space and then, after a non-linear transformation of the data, least-squares optimal curves are fitted to the surface. The fitted curves are radial functions around the basis points, exponentially decaying as a distance from the basis to the input point, and are summed linearly. In most applications, the input-output function is definitely non-linear, so linear summed curves would not work. However, the non-linear transformation converts the data from a space that is not linearly separable to one that is linearly separable, or at least might be. In addition, since the network only needs to be trained once, it is much quicker than using error back propagation.

The first network parameter for a radial basis function is the number of basis points to be used. The value for this parameter is the same as the number of nodes in the hidden layer, each one representing a basis. The number of basis points ranged from 10 to 300. The second parameter is the size of the radial functions, representing an inverse scaling factor of the exponentially decaying radial function. A small radial function means that approximations will be local; however, this will require more basis points to represent the entire curve. The size parameter ranged from 1 to 100.

The results using radial basis functions were:
- Training was very fast, on the order of a minute, but training time increased quickly with the number of basis points.
- Classification performance was not as good as with iterative back propagation, with the best at 82% correct classification of previously unseen input-output pairs.
- The optimal number of basis points was 300, with a size parameter of 4. This indicates that the curve being fitted was quite non-linear and that the basis approximations were local. This, in turn, indicates that the non-linear transformation did poorly in removing the non-linearity from the input-output space.

16

4.2.3.2 Training Methodology Implementation

Based on the results of our neural network exploration, an iterative, error back-propagation network was selected with 1 hidden layer, 30 hidden nodes, and a learning speed of 0.01. This neural network's input layer matched the population of C^2 network with 34 nodes to describe the state. It represented the message latencies with 18 output nodes corresponding to discrete time bins. Using these parameters, we then determined the number of training input-output pairs (the training set size) needed to accurately represent the state-performance relationships of the C^2 network. In terms of analytical utility of the concept, it is important to balance the training set size and accuracy; given that the original calculation of a single input-output pair could require multiple executions of very large C^2 network models, implying an expenditure of significant time and resources to produce the training set data.

For the selected neural net, the size of the training set was varied from 100 to 10,000 points. The results are presented in Figure 3, which shows the classification rate, or fraction of correctly mapped from state to performance, in the test set versus the number of points used in the training set. Here, a value of one represents perfect performance against the test set. Note that the classification rate axis starts at 0.7 rather than 0.0 to magnify the changes in slope across the range of training set sizes. Also, the values shown are averages, and vary depending on the initial weights of the network and which values were selected for training versus testing. In general, the 90% confidence intervals for the distribution of classification performance represent less than a 3% difference in the classification rate.

Figure 3: Classification rate versus training set size.

As shown in figure 3, the performance of the network levels off between 1,000 and 2,000 training set pairs. Thus, in this case no more than 2,000 training set pairs would need to be generated. This number does depend, of course, on the acceptable error in the estimate of C^2 performance.

Figure 4 shows the amount of time it took to train the back-propagation neural network. The time required to train the network is dwarfed by the amount of time it takes to generate the training set. The time it takes to evaluate a single output as a function of the input as required in the campaign simulation is very small; on the order of milliseconds. Error back-propagation and the repeated calculations of outputs dominate training time.

Figure 4: Training time for iterative error back-propagation network as a function of training set size.

4.2.4 Neural Network Integration into THUNDER

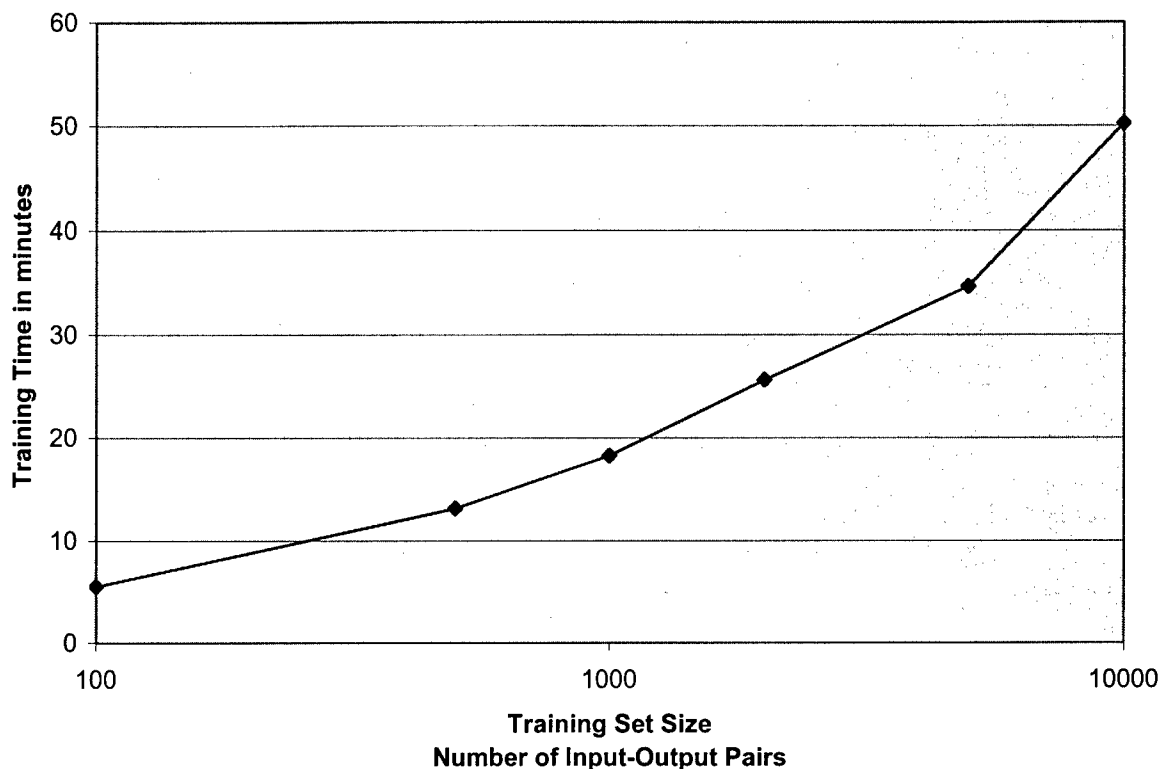The campaign model used for this effort was THUNDER version 6.702. THUNDER is owned by the Air Force Studies and Analyses Agency (AFSAA) and is a component of the Air Force Standard Analysis Toolkit. It is a stochastic, two-sided, analytical simulation of campaign-level military operations built to examine issues involving the utility and effectiveness of air and space power in a theater-level, joint warfare context. THUNDER is written in SIMSCRIPT II.5$^{®}$ and consists of approximately 350,000 lines of code contained in 1,500 routines.

The neural network code developed in this effort and described above was integrated into THUNDER. This created an experimental, unique version independent of the formal configuration-controlled THUNDER simulation. The integration consisted of three major steps:

(1) develop a C language wrapper suitable for calling from a SIMSCRIPT program in which to embed the C++ neural network class;

(2) replace Thunder's existing OTBM sensor-to-shooter latency calculation with an interface to the neural network; and

(3) compile and link resulting software components into a single binary executable.

Step(1) The C language wrapper for the neural network ("SIMSCRIPT-to-C++ Interface" as part of Figure1) was required because SIMSCRIPT does not support a direct link to C++ code. This wrapper presents a single interface to a SIMSCRIPT program consisting of a

function that is called with a vector representing the state of each node in the C^2 architecture and that returns a corresponding latency value. This wrapper function performs the following specific tasks:

(1) initialize an instance of the feed-forward neural network class from a previously generated neural network definition data file;

(2) initialize the neural network input nodes using the given state vector;

(3) update (proceed feed-forward through) the neural network;

(4) capture resulting output nodes states;

(5) look up latency value from user-input delay map file corresponding to neural network output node with the highest value; and

(6) return this latency value to the calling program.

Step(2) The "THUNDER-to-Neural Network Interface" shown as the lower, middle blue box in Figure 1 was implemented in SIMSCRIPT as an integral part of THUNDER. In general, this mediates the handling of THUNDER C^2 entities representing ground and airborne command authorities, communications facilities and sensors. This code augments the sensor-shooter latency functionality in the currently fielded simulation and performs the following specific tasks:

(1) load a user-defined C^2 entities map file which specifies the mapping between THUNDER entities with roles in the C^2 system and the corresponding input nodes in the neural network. THUNDER entities supported in this implementation are satellites, high value asset orbit points (for aircraft such as AWACS, JSTARS, etc.), and strategic targets (for surface facilities such as satellite downlink stations, intelligence processing facilities, headquarters, etc.)

(2) create the appropriate input state vector for the neural network using the C^2 entities map and the current states of the various entities in the simulation;

(3) determine OTBM mission sensor-to-shooter latency by calling the neural network. This consists of passing the neural network input state vector to the C function defined in the SIMSCRIPT-to-C++ Interface; and

(4) insert the returned latency value back into the baseline OTBM simulation algorithm .

Step(3) The SIMSCRIPT code was then compiled to object code using a SIMSCRIPT compiler and, the C and C++ code were compiled using the g++ compiler. The resulting object codes were then linked into a single executable. This required the manual inclusion of all the runtime and system libraries needed by SIMSCRIPT, C, and C++.

4.2.5 Demonstration Application

We performed an experiment to evaluate this neural network/THUNDER integration in terms of its practical performance and its analytical utility. The experiment took its form from designs and approaches typically associated with analytical studies conducted throughout the Air Force M&S community.[8] We used a scenario database reflecting a major theater conflict consistent with Defense Planning Guidance to establish a scenario environment to guide the simulation of military operations within THUNDER. The scenario involved the deployment and use of large numbers of joint military forces equipped with a wide range of systems and assets across a battlespace extensive in time and geography.

The initial run matrix comprised four cases generated by changing two variables in the scenario – differences in warning time (promoting a variance in the availability of C^2 architecture assets and the intelligence preparation of the battlespace) and differences in threat intensity (numbers of TBMs). In the long warning cases (W+), the flow of materiel into the theater provided earlier deployment of C^2 assets. This gave a more complete architecture with higher density of sensors, processing and communications nodes early in scenario time. In the short warning cases (W-) represent a higher degree of surprise, with the exercise of contingency deployment that result in a later full population of the C^2 network in scenario time. The TBM cases present low (T-) and high (T+) cases for the number of TBM units, missiles and warheads the opposing force fields in the scenario. The T- cases assume a lower ability for the threat nation to augment its TBM forces. The T+ case assumes less effectiveness for diplomatic, economic, and other non-military efforts to curtail the proliferation of the TBM threat. Finally, we explicitly eliminated a C^2 node from the architecture in the N- excursion. Specifically, we withheld a key airborne sensor key to the shortest latency path. This provided a direct, constant change in the C2 architecture that we mapped to changes in the sensor-shooter performance and OTBM mission effectiveness. Operationally, this "missing node" realistically maps to experience with High-Demand/Low-Density Intelligence, Surveillance and Reconnaissance assets where special platforms are not always available to the commander. Forty days of operations were simulated. For each case, 10 replications were run to provide for statistical stability given the stochastic nature of the THUNDER simulation. Upon evaluation of the results of this initial matrix, an additional excursion was run involving a third variable – the nullification of a key node in the C^2 architecture. Each of the five cases was run in both the THUNDER 6.7 baseline version and the integrated neural network version of THUNDER. Our assessment of the experiment is discussed below. Table 2 provides a legend for the charts in the discussion. Data on the charts has been normalized.

Case	TBM Threat	Warning	Node
T-W+	low	long	
T-W-	low	short	
T+W+	high	long	
T+W-	high	short	
T+W+N-	high	long	missing

Table 2: Experiment Case Nomenclature and Descriptions

4.2.5.1 Feasibility Issues

We used the experiment to evaluate the neural network performance from two different feasibility perspectives. First, to be successful as an abstraction mechanism, the neural network needed to enable the host simulation to be sensitive to state changes in the C^2 network architecture. Second, to be successful as a software implementation, the neural network had to function as an integrated part of the host simulation without imposing unacceptable operating

conditions. Each of these perspectives regarding the practicality of the integrated neural network / THUNDER tool is discussed below.

As illustrated in Figure 5, the neural network successfully supplied THUNDER with a range of latency values associated with variations in the state of the C^2 network.
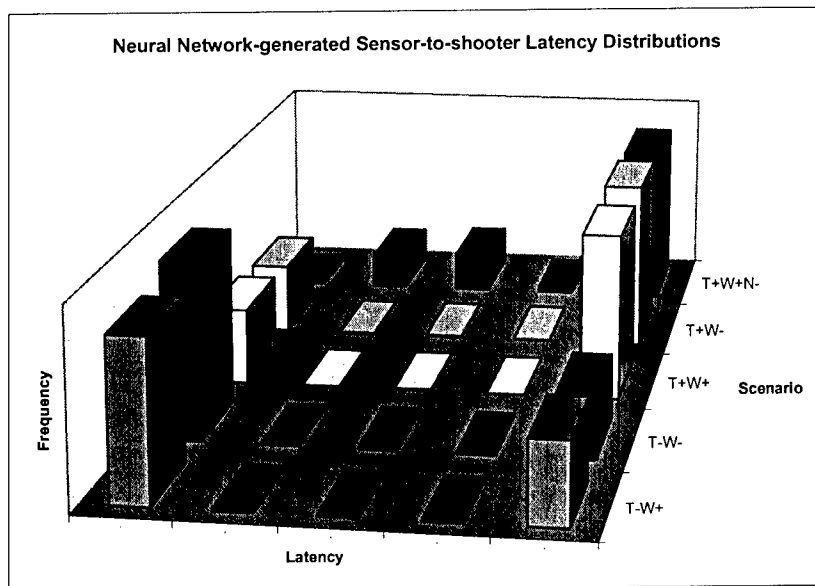

Figure 5: C^2 Network Architecture Sensitivities in Neural Network-generated Latencies

The distribution of latencies generated by the neural network was more sensitive to threat level than warning time. In the excursion case with the neural network, removal of one key node resulted in a markedly different latency distribution. In contrast, the baseline model exhibited no sensitivity in this case. In general, the primary driver of architecture state changes between the cases resulted from harassment of friendly high-value assets, forcing them to retrograde and consequently be temporarily unavailable to the C^2 network architecture.

The experiment demonstrated the success of the integrated neural network/THUNDER tool as a software implementation. The integration did not compromise the operation of the THUNDER simulation (e.g., by provoking fatal errors, infinite loops, etc. in terms of software incompatibilities. This provides evidence of the ability of the neural network to satisfy the data availability demand imposed by the design of the THUNDER simulation runtime architecture. The trained Neural Network provided the mapping between the C^2 network state and the sensor-shooter information delay for target detections numbering in the hundreds. We implemented output of the detailed transactions across the THUNDER-to-NN Interface to provide confirmation of the data exchange and visibility into the content.

Further, the experiment demonstrated that, given the availability of a suitable training data set, the neural network could provide more insight into cause-effect relationships in the C^2 domain than the methodology in the currently fielded THUNDER. The neural network provided traceable, experientially derived sensor-shooter performance values compatible with timeframes

associated with a typical campaign-level analytical environment. THUNDER simulations normally involve several minutes of CPU time per simulated day depending on the duration and complexity of the scenario. To compute the amount of CPU time required for a given simulation-supported analytical study, this individual run duration must be multiplied by the number of simulation replications required for statistical stability and by the number of cases and excursions in the run matrix. Significant increases in runtime attributable to the incorporation of the neural network could compromise its practical utility in terms of the typical study design found in formal Analysis of Alternatives and force mix analyses. The experiment revealed no significant runtime differences between the integrated neural network/THUNDER case and the THUNDER 6.7 baseline case. While the implemented neural network is an efficient algorithm, the design and constraints of the experiment also played role in this result. Considering the large number of interactions associated with the operation of THUNDER, the OTBM focus precipitated a relatively small number of invocations of the neural network by the core simulation. Across the various cases explored, invocations of the neural network ranged from 13,000 to 94,000 per replication.

4.2.5.2 Utility Issues

We also used the experiment to evaluate the analytical utility of the integrated neural network/THUNDER tool. Again, we approached the issue from two perspectives. First, we evaluated the tool's ability to facilitate the use of credible C^2 source data in the context of a larger simulation. Second, having introduced these credible effects more directly and rapidly by means of the neural network, we examined the results of the experiment for any noticeable impact in terms of potential analytical use for campaign simulation.

The principal analytical benefit envisioned for the neural network abstraction tools is to enhance the credibility that comes from being able to provide a simulation with higher quality inputs; in this particular instance, for C^2 performance as a function of architecture/network state changes that are clearly traceable to empirically-derived values. While to some degree this has been possible in the past, that method was one of calibration. In contrast, the neural network enables a more direct "injection" method that is clearly more traceable and has the potential to be significantly faster. THUNDER version 6.7 is sensitive to architecture/network state changes only through user-defined values. The analyst populates the THUNDER algorithms with data establishing the latency or delay associated with a given state using statistical distribution functions and rule language code. Given enough resources, time and effort, an experienced THUNDER analyst could conceivably determine the appropriate distributions to capture the behavior of a C^2 network as indicated by higher-resolution models. As shown in Figure 6, however, this would have to be done iteratively; requiring multiple cycles of sequential steps. After securing and evaluating the results of detailed models to determine behaviors appropriate to THUNDER's more aggregated context and level of abstraction, the analyst would need to establish an initial set of inputs to the pertinent data files, run the simulation, assess the results against the expected behavior and refine the inputs until desired behavior is obtained.

23

Traditional THUNDER Calibration Process

Initiate Study ➡

(subsequent iterations as required)

Iteration 3 ⬆

- *Evaluate and determine aggregate behaviors*
- *Develop appropriate input data*
(values, statistical distributions, rules, etc.)
- *Run simulation*
- *Assess results against expected outcomes*
- *Refine input data*

"Weak" C² Thread
- Less traceable to source data

Iteration 2 ⬆

- *Evaluate and determine aggregate behaviors*
- *Develop appropriate input data*
(values, statistical distributions, rules, etc.)
- *Run simulation*
- *Assess results against expected outcomes*
- *Refine input data*

**Detailed Models /
Operational Experience**

Iteration 1 ⬆

- *Evaluate and determine aggregate behaviors*
- *Develop appropriate input data*
(values, statistical distributions, rules, etc.)
- *Run simulation*
- *Assess results against expected outcomes*
- *Refine input data*

**Credible
C² Data
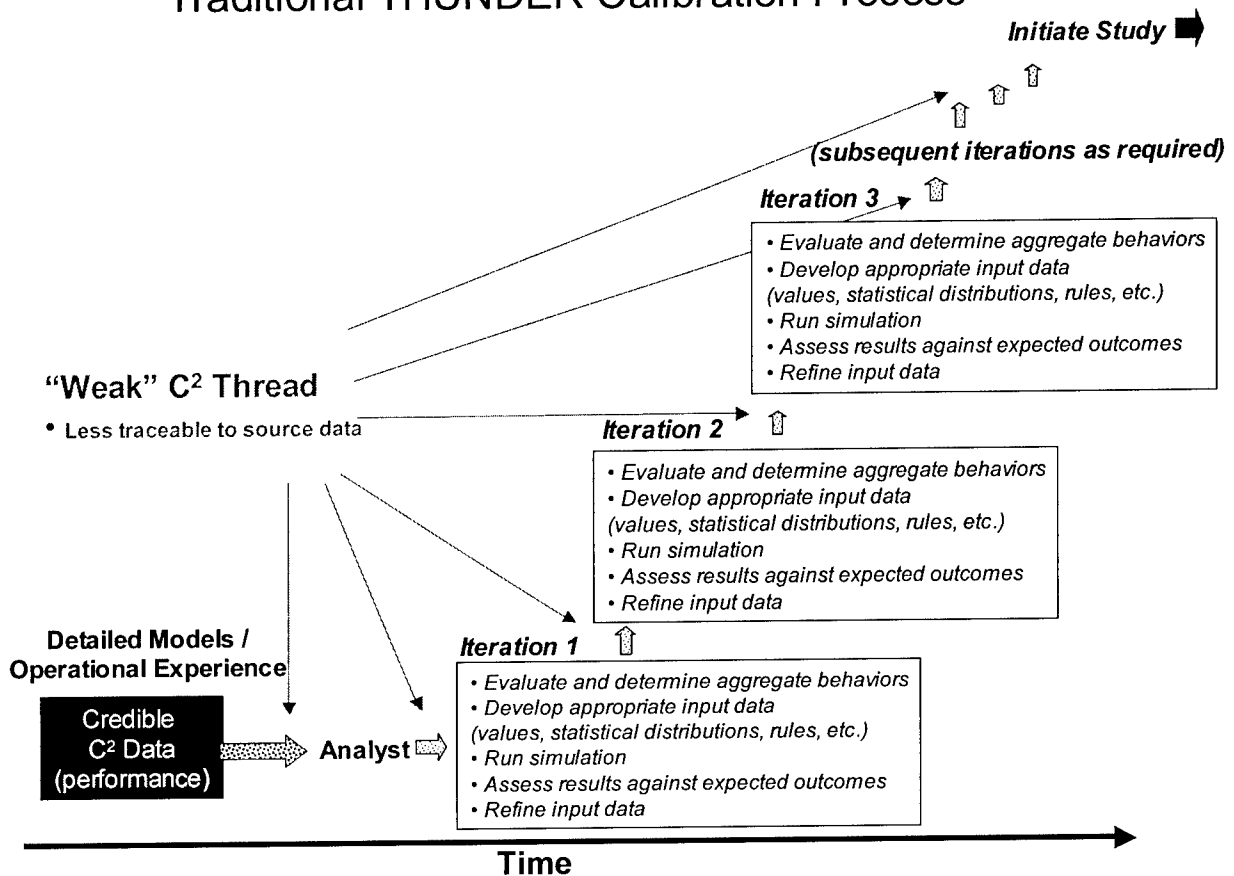(performance)** ➡ **Analyst** ➡

Time →

Figure 6: Traditional calibration approaches impose significant barriers to traceability and large penalties in time.

The time required to rigorously calibrate THUNDER in this fashion is rarely available in practice. In addition to the amount of time and effort needed, the process is relatively opaque and requires a broad base of expertise (subject area, source model, host model, statistics, etc.) on the part of the analyst.

The neural network, in contrast, was intended to automatically support the existing sensor-shooter delay logic by drawing from a range of values as reflected by data derived from operational experience or detailed models. Within the process described in Figure 7, using a properly trained neural network to provide inputs to a simulation – given the availability of suitable data for training – requires significantly less time and effort than the calibration process. In addition to reducing the amount of time spent initiating a study, the neural network also provides a more traceable derivation trail or thread between the original source data and C^2 effects injected into the simulation. Further, the strength of this thread is a function of the quality of the source data and the accuracy of the neural network. In contrast, manual calibration relies on the knowledge and skills of the analyst to aggregate C^2 behaviors from detailed results and mathematically induce appropriate effects in the host simulation.

Integrated Neural Network / THUNDER C² Injection Process

"Strong" C² Thread

- Directly traceable to source data

Initiate Study ➡

Train Neural Network

• *Select appropriate neural network configuration*
• *Identify optimal training methodology*

Detailed Models / Operational Experience

Credible C² Data (performance) ➡ **Develop Training Data Set**

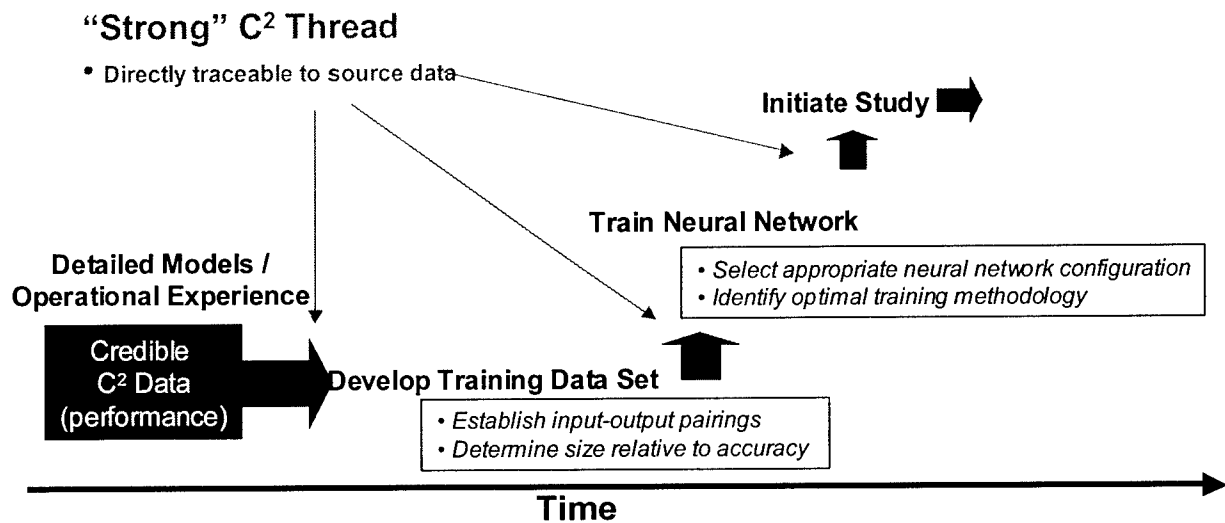| |
|---|
| • *Establish input-output pairings* |
| • *Determine size relative to accuracy* |

Time ➡

Figure 7: Using the neural network to generate C² latencies has the potential to save time even as it provides greater traceability and credibility.

It is important to note the caveat "given the availability of suitable data for training" in the preceding paragraph. As mentioned previously, the scope of the funded effort precluded the collection or production of a training data set from the output of detailed models reflecting the performance of "real-world" C² architectures as a function of state conditions. This would have involved many excursions of a complex network architecture model, with associated data post-processing and analysis to arrive at a training set of the requisite quality for actual analytical use. For this effort, we generated an illustrative training set upon which to train the neural network. While satisfactory for a proof-of-concept demonstration, development of this NN training set certainly required less time and resources than a more analytically rigorous training data set would. As illustrated in Figure 8, creation of a NN training data set is a major resource consideration with respect to campaign simulation. Development time can expand markedly if the data or detailed model results are not readily available but must instead be collected or produced.

Integrated Neural Network / THUNDER C^2 Injection Process

Unresolved Issue

• Time/resources required to acquire or produce data for training set

Initiate Study ➡

Train Neural Network

Detailed Models / Operational Experience

• *Select appropriate neural network configuration*
• *Identify optimal training methodology*

Credible C^2 Data (performance) ➡ Develop Training Data Set ⬆ ⬆ ⬆ ... ?

• *Establish input-output pairings*
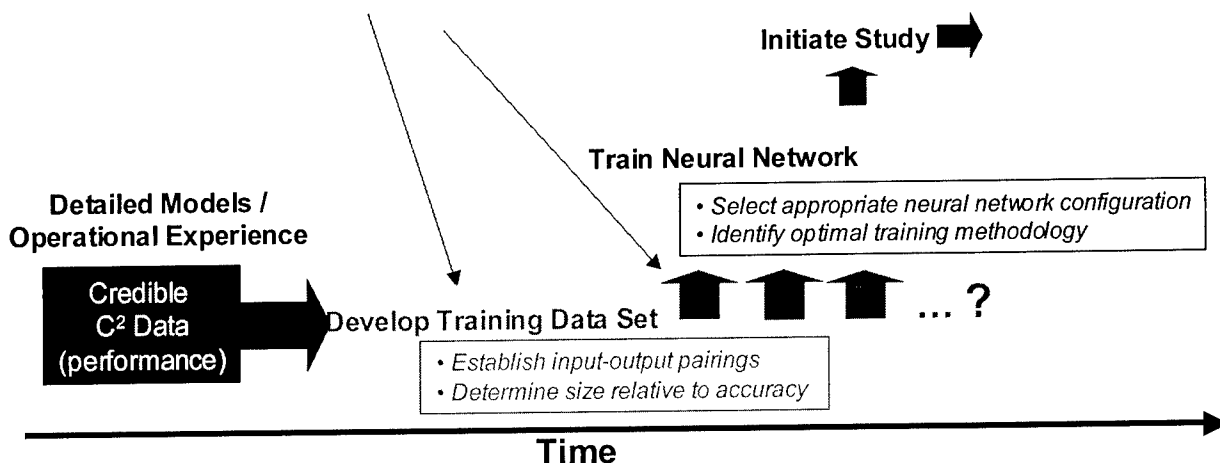• *Determine size relative to accuracy*

Time ➡

Figure 8: The time and effort required to acquire or generate an analytically rigorous training data set could be substantial.

We also examined the results of the experiment for any noticeable impact in terms of potential analytical use for campaign simulation. Campaign analysis typically relies on an analytic concept of traceable chain of causality, or "thread," to map cause to effect. Employing an OTBM thread in this experiment, we would expect to see variations in the state of the C^2 network architecture; latencies associated with state changes; TBM kills; launches of TBMs; and the effects of TBM warheads on targets. As previously discussed, this integrated neural network / THUNDER tool successfully generated C^2 network architecture state changes and corresponding latencies; providing depth for further analytical exploration. Continuing to follow the thread concept, we investigated cascading effects by looking at some typical campaign measures of effectiveness. As shown in Figure 9, comparing the baseline methodology and the neural network, we observed case-dependent differences in kills of TBM transporter/erector/launchers (TELs). All neural network cases showed fewer TEL kills than the corresponding baseline cases, reflecting longer average latencies generated by the neural network as compared to the baseline methodology. In general, the differences between cases were driven more by the number of TELs than by any sensor-to-shooter latency effects. While this masking effect makes it difficult to see the neural network's sensitivity to C^2 architecture changes in this particular instance, we postulate that this metric could reflect greater sensitivity given different C^2 architecture and scenario assumptions.

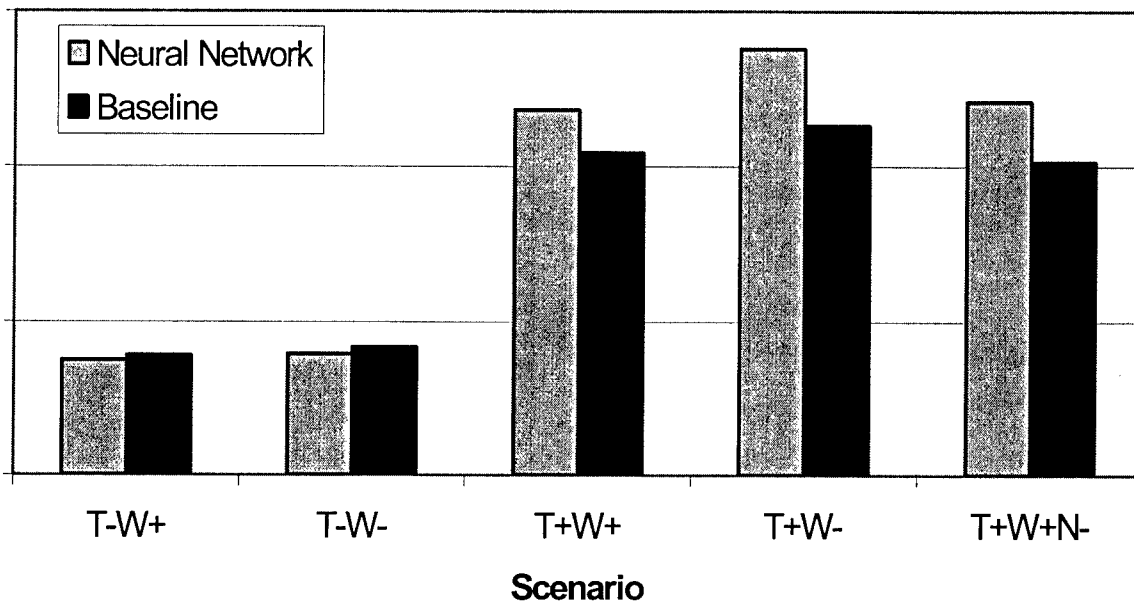## TBM Launches -- 1st 10 Days

Figure 9: Comparative TBM TEL Kills

Figure 10 illustrates a different metric for assessing TEL kills. It reflects the ratio of TELs killed by OTBM missions before missile launch, clearly an important consideration in evaluating mission effectiveness. From this perspective, the neural network showed considerable sensitivity to C^2 network architecture changes while the baseline was insensitive. Consider the differences between the W+ and W- cases. Here, differences in the C^2 architecture result from the deployment time-line for the C^2 nodes. The baseline THUNDER results are insensitive to these changes. The neural network, trained to translate the architecture change into differences in sensor-shooter performance resolves more dramatic advantages for attack operations against launchers in the pre-launch phase of their target life-cycle for the longer warning cases. This implies that the baseline simulation does not resolve the changes in the sensor-shooter kill chain attributable to the deployment of C^2 assets to the degree provided by the neural network abstraction tool. Our analytic thread relies on how small differences in sensor-to-shooter-latency can cause large differences in OTBM mission effectiveness.

Continuing with the campaign analysis thread concept, Figure 11 shows the number of opposing force TBMs actually launched. While some of the differences between cases are attributable to differences in density of threat missile force, the neural network cases are clearly more sensitive to C^2 network architecture changes the ability of OTBM strike aircraft to successfully find and attack these time-critical targets.

By following the campaign thread in this manner, we were able to directly link enemy TBM launches to friendly C^2 network architecture states. The experimental scenario was not intended to support real-world acquisition or force mix decisions with in-depth analysis but it does provide a proof-of-concept. In a scaled up version/implementation, this thread could be expanded to contain effects of launched TBMs and resulting impacts on campaign outcome.
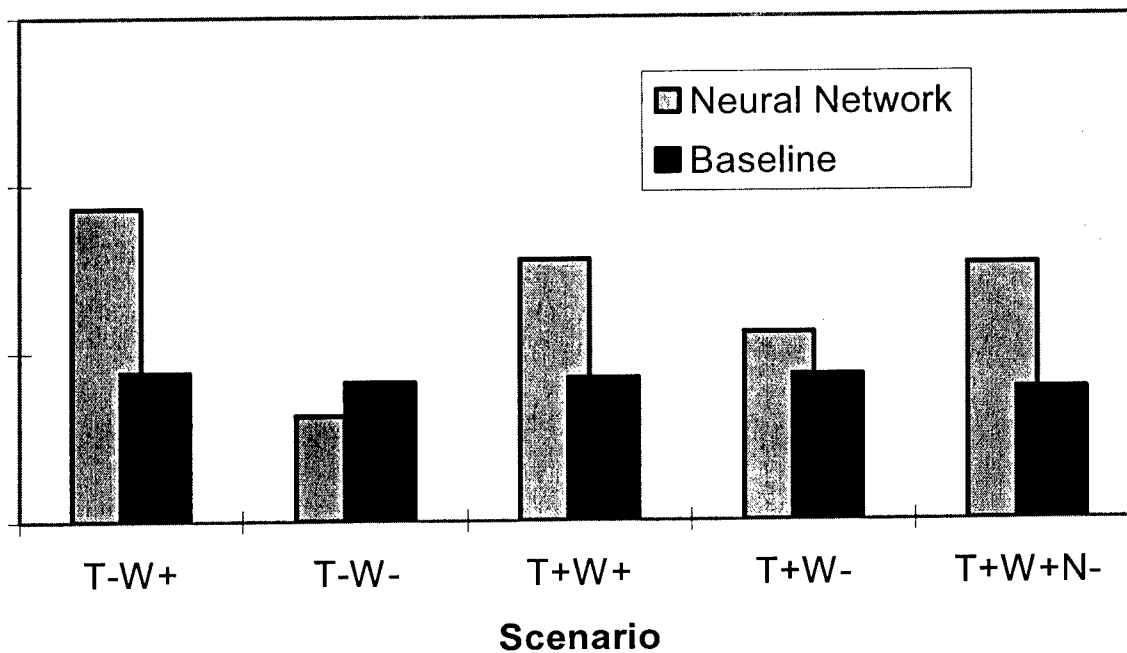
27

% Prelaunch TEL Kills

Figure 10: Comparative TBM Pre-launch Kills
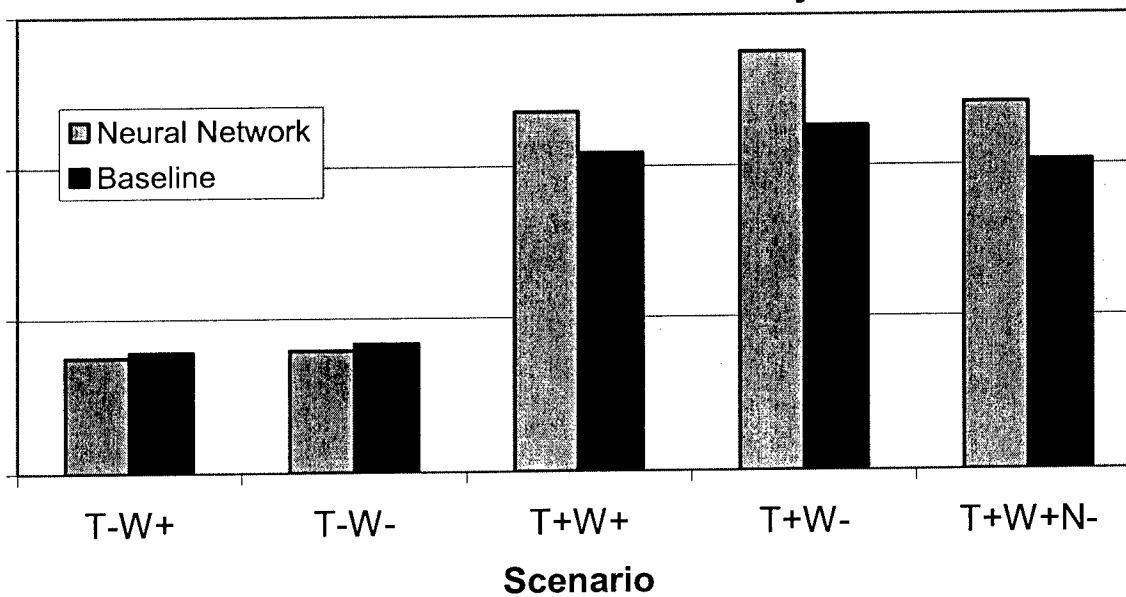
TBM Launches -- 1st 10 Days

Figure 11: Enemy TBM Launches

5.0 Conclusions

The project successfully explored the development and application of a particular model abstraction application. This effort served as a proof-of-concept that a neural network can map cause and effect relationships automatically within THUNDER instead of having this performed manually by a subject matter expert manipulating THUNDER data. We demonstrated that a specific type of neural network could be trained to an acceptable level of accuracy within the bounds of analytical rigor associated with conventional campaign analysis. Further, we demonstrated that a particular neural network implementation could be incorporated and responsive to the timeframes required by a campaign-level simulation. We demonstrated that a properly trained neural network could quickly provide a simulation with an increase in richness of the C2 relationships over the aggregate representation currently implemented. The abstraction technology provided a diverse range performance values related to network state information and traceable to credible sources. Finally, our demonstration showed that this range of values would result in perceptible differences in aspects of military effectiveness that could have implications for campaign-level analysis.

The project demonstrated that neural networks as a model abstraction technology offer utility in exploring a very specific yet important trade-space. Specifically, in the application of C2 concepts to counter the threat posed by proliferating TBM technology. Studies of alternatives in this mission area are currently in the forefront of in military operations and acquisition planning as illustrated by Aerospace Command, Control, Intelligence, Surveillance and Reconnaissance Center (ASC2ISRC) and ongoing Electronic Systems Center (ESC) Time Critical Targeting Cell acquisition.[9] However, it is not clear how this potential can be fully exploited in the near term, given that the project was limited in scope. We recommend further research addressing a larger scope and scale of effort necessary to produce a more comprehensive, analytically rigorous training data sets, neural network architectures, and training methods appropriate to different applications. Future efforts should explore availability of credible C^2 source data; specifically, the possibility of obtaining information reflecting real-world operational experience as well as time and resources necessary to produce detailed model results from existing network models and tools. This could conceivably exploit developments in C^2 network simulation, specifically, the NETWARS effort in OSD/C^3I[10] and the Model Reference Toolkit program at USAF Electronic Systems Center.[11] Questions concerning the level of effort required to acquire or produce an analytically rigorous training data set for C^2 effects (as opposed to an illustrative set) must be explored in order to make a further judgment of the feasibility and utility of the neural network as a model abstraction technology.

6.0 Lessons Learned/Extensions

A primary concern derived from our experience involves volume of training data required in application of the abstraction method. Should this volume prove unmanageable in terms of building new training sets, a mitigating approach might be to develop a "superset" construct that essentially combines multiple excursions to compress the training requirements. This approach

would include a single training set describing a number of architecture alternatives, for example: distributed decision-making; new system nodes; differing CONOPS and associated rules; and performance parameters. The training data would encompass both the baseline and excursion data. To extend the TBM example used in this effort, we might pursue an analysis of alternatives for Moving Target Indicator (MTI) sensors to detect the TEL targets. Here, we could build a "superset" architecture that included all the alternatives, say Joint STARS, Space-Based Radar (SBR) and an Unmanned Aerial Vehicle (UAV) with an MTI sensor payload. We could train the neural network on comprehensive states of this super-architecture then implement specific alternatives by explicitly limiting the states of the different MTI options alone or in combination. Here, we could "switch off" all but the Joint STARS for a baseline case, then set explicit states for the UAV, SBR or combinations of the options. In effect, the neural network would abstract a related suite of models of related architectures. This would address the problem by training the network on the union of the excursion training sets. This provides a single neural network encoding the behavior of multiple excursions.

The project also revealed areas for further refinement and exploration of the integrated neural network/THUNDER tool's potential to promote analysis of the campaign-level implications of C^2 effects. As a proof-of-concept effort, the project was naturally limited in its focus, involving the neural network and its approximation capabilities in a single area or thread of a complex simulation. Whereas this project examined C^2 performance as factor in the OTBM mission within THUNDER, other areas, particularly missions associated with attacking integrated air defense systems (IADS), seem ripe with potential. In addition, it would be useful to further explore the analytical depth afforded by the range of values available from the neural network. Research should be considered that manipulates scenario variables to investigate how and where to expand the sensitivity of the simulation to architecture state changes.

An additional area for exploration involves continuous variable problems. For this effort, we employed the neural network in a pattern-matching mode. In contrast, continuous variable problems involve approximating complicated non-linear solution surfaces. Potential places to focus on include probabilistic phenomena. In C^2 networks, this could address problems in dynamic bandwidth allocation, abstract queuing in intelligence broadcasts and the related queue management algorithms that prioritize information across tiers of urgency.

7.0 References

[1] R. Richards, "MORS Workshop On Analyzing C4ISR in 2010", *PHALANX*, **32,** pp. 10-13, 1999.

[2] S. Haykin. *Neural Networks: A Comprehensive Foundation*, Macmillan, Englewood Cliffs, NJ, 1994.

[3] *Combat Air Forces Concept of Operations for Command and Control against Time Critical Targets*, HQ ACC/DRAW, Langley AFB VA, 8 July 1997

[4] *Doctrine for Joint Theater Missile Defense,* Joint Pub. 3-01.5. 22 February 1996.

[5] *C4ISR Architecture Framework, Version 2.0,* DoD C4ISR Architectures Working Group, 18 December 1997.

[6] S. Even, *Graph Algorithms.* Computer Science Press, Inc., Rockville, MD, 1979.

[7] J. Hertz, A. Krough and R.G. Palmer, *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA. 1991

[8] SAS-002, *NATO Code of Best Practice for Command and Control Assessment* (Draft) AC/243(Panel 7)TR/8, 1998.

[9] *Operational Requirements Document for Time Critical Targeting (TCT) Functionality,* AC2ISRC (USAF) 401-98[DRAFT], AC2ISRC/C2N, Langley AFB VA , 20 June 00.

[10] Stubbs, B. *NETWARS - An Abridged History and Overview,* OASD/C3I)/PAI/DSC Briefing presented at OPNETWORK 2000, Washington, DC, 31 August, 2000

[11] Gibson, R. "Integrating Operational and Systems Architectures: How Modeling and Simulation Can Empower Command and Control Acquisition," *PHALANX,* **32,** pp. 14-15,31.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science

and Technology to meet Air Force unique requirements for

Information Dominance and its transition to aerospace systems to

meet Air Force needs.